Powering Institutional Research with R

Thomas Jay Benjamin Lynzee Murray

2023-10-19

Table of contents

Preface

This online reference guide has been developed to accompany the October 19, 2023 workshop at the Ohio Association for Institutional Research and Planning.

The workshop will follow a timeline aligned to the sections of this reference guide:

- Chapter 1: Software setup and overview
- Chapter 2: Importing and cleaning data
- Chapter 3: Statistical analyses & data visualizations
- Chapter 4: Building parameterized reports
- Chapter 5: Collaborating on code

This reference guide is designed **for institutional researchers**, **by institutional researchers**, so it's chock full of examples relevant to IR and often compares R functions to what you might use in Microsoft Excel.

1 Software setup and overview

1.1 Installation

First, we'll want to install the R software. Go to https://r-project.org and follow the **download** link.

That will take you to the **Comprehensive R Archive Network**, or CRAN, which is "a network of servers that store identical, up-to-date, versions of code and documentation for R."

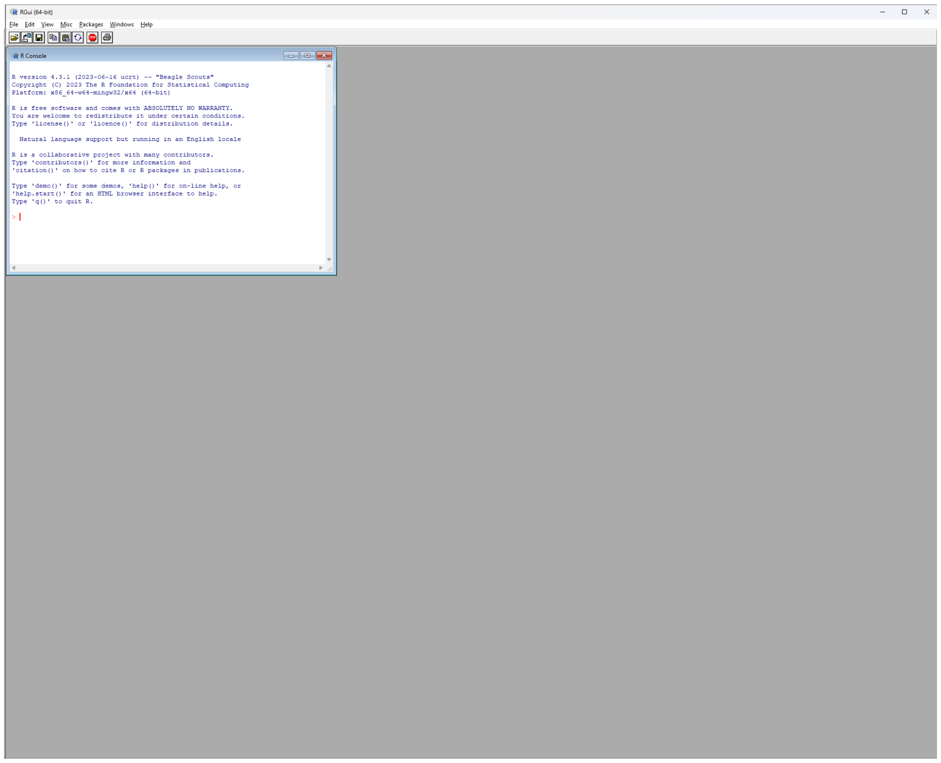
The closest CRAN mirror for Institutional Researchers working in Ohio is hosted by Case Western Reserve University. Its web address is https://cran.case.edu.

Choose the distribution appropriate for your operating system under "**Download and Install R**," then click on the link for "**install R for the first time**," and then finally the larger "**Download R...**" link. Accept the licenses and the default settings.

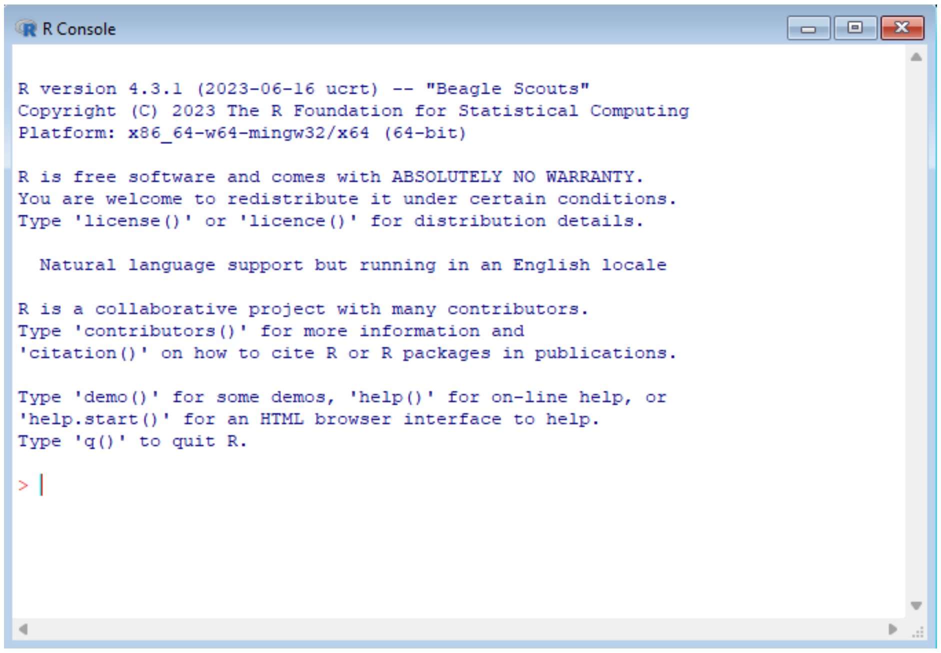
1.2 Using the R Terminal

You should now be able to find R listed among your installed programs. Go ahead and run it. You should see a screen like this:

The inner window is the **R Console**, also called the **R Terminal**, which is the R software interface:



```
R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```



```
R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Let's run our first command. Since R is statistical software after all, let's use it as a calculator. Enter `1 + 1` and press `Enter`. You should see the following:

```r
1 + 1
```

```
[1] 2
```

Congrats, you've run your first R command!

This way to access R is fairly limited, so we can close out of the program (or enter the `q()` command). R will ask you if you'd like to save the workspace - just click "No".

1.3 Using an IDE

1.4 Installing R packages

While base R contains all kinds of helpful functions and tools, installing R packages provide additional functionality.

CRAN includes both the base R system and an array of approved R packages.

One such package that we'll be using is `tidyverse`, a collection of R packages that make data analysis in R better. We'll learn more about tidyverse in Chapter 2.

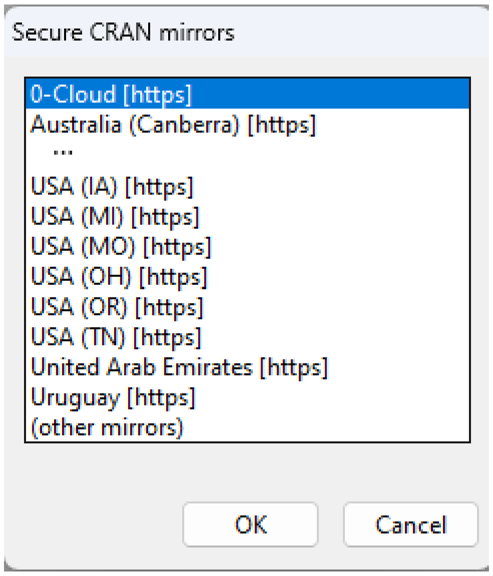
To install the package, run the following command in the terminal:

```r
install.packages("tidyverse")
```

! Important

Note that the package name must be in quotes (`""`) in the `install.packages()` command.

When you first install an R package in a session, you will be prompted to choose a CRAN mirror, like you did when you first downloaded R:


Secure CRAN mirrors

0-Cloud [https]
Australia (Canberra) [https]
...
USA (IA) [https]
USA (MI) [https]
USA (MO) [https]
USA (OH) [https]
USA (OR) [https]
USA (TN) [https]
United Arab Emirates [https]
Uruguay [https]
(other mirrors)

We'll choose `USA (OH) [https]`, which is the mirror hosted in Ohio by Case.

Tip

See Section 1.5.2.1 for details on how to set your default CRAN repository.

We'll install additional packages later.

1.5 Writing an R script

Up until now, we've entered commands directly into the R terminal. But much of the power of using R comes from writing many lines of code that work together.

To do this, we can create files that contain such code, as **scripts**.

Create a new `.R` file in your project folder. You can call it anything you'd like, such as `script.R`. By using the `.R` (or `.r`) extension, you're indicating that the file is an R script.

At the top of the file we want to load any R packages that we'll be using. We do this with the `library()` function. Let's load `tidyverse`, then run our code:

Listing 1.1 `script.R`

```r
library("tidyverse")
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts(
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all c
```

Notice that the output shows the exact packages loaded as part of `tidyverse`. Not all packages have an output, but many do display some helpful information. The conflicts section notes that certain functionality from base R (`stats`) is masked by the packages loaded. This means when we run a function like `filter()`, by default it will now use the version from `dplyr`, which is part of `tidyverse`. We can always explicitly run a certain version by entering a function in the format `package::function()`, like `stats::filter()`.

⚠ Warning

Note that we never include an `install.packages()` command in our R script, only in the terminal. We don't want to modify our computer using our script, and this is espeically important when we get to sharing code!

Now we can add additional code to the script. Let's create a new object `my_fav_number` and assign it the value of `2`. Don't worry, we'll go over exactly what is happening here in Chapter 2.

Listing 1.2 `script.R`

```r
library("tidyverse")

my_fav_number <- 2
```

Finally, let's add notes to ourselves so that we can remember what our code is doing. To do this, we use **comments**. In R, comments are denoted by the `#` symbol. When R encounters a `#` symbol, everything in the same line after that symbol is ignored when running the code. We always add a space between the `#` symbol and the text we'd like to use as a comment.

We can use comments at the end of a line of code to describe what is happening in that line, and we can write a comment as a whole line in the code to show what is happening in sections of code:

Listing 1.3 `script.R`

```r
# load libraries
library("tidyverse")

my_fav_number <- 2  # 2 is my favorite number
```

Now that we've written our first R script (**!**), we can run it. We can either run the entire script at once, or walk through the code line-by-line with `Shift-Ctrl`. Walking through code line-by-line can be a great way to develop code and test as you go.

A few notes about keeping your code clean:

- Use **blank lines** to separate logically separate lines of code.
- Use **spaces** properly around characters and symbols.
- Use **comments** to keep your code comprehensible.

We'll go over additional notes about writing clean code in the coming chapters.

Exersices

1.5.1 Exercise 1

In Chapter 2 we'll use the package `readxl` to read in data from Excel files, since IR professionals often encounter data we need to work with in Excel files! How would you install the `readxl` package?

```r
install.packages("readxl")
```

1.5.2 Exercise 2

Start a new R script that loads the `readxl` package. Include a comment to remind you of what is happening.

```r
# load libraries
library("readxl") # to read data from Excel files
```

Extra: Keeping R up-to-date

1.5.1 Updating R packages

In the terminal, run:

```r
update.packages()
```

Packages with new updates will be detected and you will be prompted to respond `Y` or `yes` to accept.

If you want all packages updated without your confirmation, you can add the `ask = FALSE` argument:

```r
update.packages(ask = FALSE)
```

Tip

You will see a warning when packages are built under a different version of R than you are running. If you see this, you likely want to update your version of R.

1.5.2 Updating R

The `installr` package can assist with keeping the R installation up-to-date. In the terminal, run the following:

```r
installr::updateR()
```

Tip

You will be prompted as to whether you'd like to copy over all packages from the current version of R. This is a good idea!

R Profile and Environment

R will read certain options from special files that you can modify:

1.5.2.1 R profile

Your R *profile* can contain a range of settings to customize how you interact with R.

The `usethis` package can be used to edit your R profile:

```r
usethis::edit_r_profile()
```

A new window will open with your R profile file that you can edit.

For example, you may want to set your default CRAN mirror to the one hosted at Case, since it is in Ohio and you may be as well. Add the following line to your R profile file:

```r
options(repos=c(CRAN="https://cran.case.edu/"))
```

When done, save the file, then close all R terminals and reopen them.

1.5.2.2 R environment

The R *environment* can store variables that you can call upon using special R functions. It is a great way to keep keys and other secrets out of your code but still accessible.

The `usethis` package can be used to easily edit your R environment:

```
usethis::edit_r_envrion()
```

To store a variable named `test_var`, add the following to the file:

```
test_var:"this is my test value"
```

When done, save the file, then close all R terminals and reopen them.

You'll be able to access such variables like so:

```
Sys.getenv("test_var")
```

```
[1] "this is my test value"
```

Warning

Some packages request that you store things like API keys as specified environment variables. You can also use this to store common URLs, including FTP sites, but keep in mind that they are accessible directly in the `.Renviron` file on your computer.

2 Importing and cleaning data

2.1 Creating R objects

2.2 Importing data

2.2.1 From a local file

2.2.2 From an online file

2.2.3 From a database

2.3 Tidy data

Exercises

Extra: Exporting data

3 Statistical analysis & data visualizations

4 Building parameterized reports

This will require the `rmarkdown` and `knitr` packages.

5 Collaborating on Code

5.1 Setting up Git

```
git config --global user.email "you@example.com"

git config --global user.name "Your Name"
```

References